Proyecto 02 – DSA para Downscaling por Interpolación Bilineal Paralela

Avance 1: Planificación y Definición Inicial

Gabriel Guzmán Rojas

Sebastián Hernández Bonilla

José Vargas Torres

31 de octubre de 2025

Índice

1. Introducción **2**

2. Objetivos específicos **2**

3. Extracción de Requisitos de la Especificación **2**
- 3.1. Requisitos Funcionales (RF) . . . 2
- 3.2. Requisitos No Funcionales (RNF) . . . 3
- 3.3. Matriz de Requisitos . . . 4

4. Marco conceptual **4**
- 4.1. Interpolación bilineal . . . 4
- 4.2. Paralelismo SIMD (DLP) . . . 5

5. Diagrama de Arquitectura **5**

6. Decisiones numéricas (Q8.8) **5**

7. Plan de verificación (borrador) **5**

8. Algoritmo de alto nivel de downscaling **6**

9. Conclusiones del avance **7**

Cuadro 1: Roles y responsabilidades

Integrante	**Rol**	**Responsabilidades clave**
Sebastián Hernández Bonilla	Arquitectura HW	Bloques SV, pipeline, temporización, BRAM/DSP.
Gabriel Guzmán Rojas	Comunicación & SW	JTAG/UART host, oráculo Python/C, *scripts*.
José Vargas Torres	Verificación	Testbenches, plan de pruebas, *golden model*.

1. Introducción

Este avance establece la planificación inicial para diseñar e implementar una **arquitectura de dominio específico (DSA)** en FPGA que realice *downscaling* de imágenes en escala de grises mediante **interpolación bilineal** con paralelismo **SIMD** (*data-level parallelism*). Se incluyen: objetivos, roles, extracción de requisitos, marco conceptual (bilineal y SIMD), diagrama de bloques de primer nivel y un algoritmo de alto nivel que servirá como oráculo de validación.

2. Objetivos específicos

1. Revisar la especificación oficial y extraer requisitos funcionales y no funcionales.
2. Seleccionar representación numérica fija (propuesta: Q8.8) coherente entre HW y SW.
3. Bosquejar la arquitectura modular y el *pipeline* de primer nivel.
4. Implementar un algoritmo de referencia (Python/C) para comparación bit a bit.
5. Definir plan de verificación inicial y responsabilidades del equipo.

3. Extracción de Requisitos de la Especificación

Uno de los aspectos más importantes de esta primera etapa es identificar los requisitos que surgen de la especificación del proyecto *Diseño e Implementación de una Arquitectura de Dominio Específico (DSA) para Downscaling de Imágenes mediante Interpolación Bilineal Paralela*. A continuación, se presentan los requisitos organizados en dos categorías: funcionales y no funcionales, seguidos de una matriz de trazabilidad.

3.1. Requisitos Funcionales (RF)

- **RF1 – Interpolación bilineal:** El sistema debe implementar reducción de imágenes (*downscaling*) en escala de grises utilizando interpolación bilineal, procesando píxeles de 8 bits.
- **RF2 – Factor de escala configurable:** Debe soportar un factor de escala $s \in [0{,}5, 1{,}0]$ con pasos de 0.05, modificable mediante un registro de control.

- **RF3 – Modo de operación:** El diseño debe permitir dos modos: secuencial (1 px/-ciclo) y SIMD paralelo ($N \geq 4$ px/ciclo).
- **RF4 – Control e interfaz:** Debe incluir registros de configuración y estado (busy, ready, error), y permitir comunicación con PC mediante JTAG o UART.
- **RF5 – Representación numérica:** Todas las operaciones deben realizarse en punto fijo Q8.8 para garantizar equivalencia bit a bit con el modelo C/C++.
- **RF6 – Arquitectura modular:** El sistema debe dividirse en módulos sintetizables: Control FSM, Loader, Register File SIMD, Unidad Bilineal, y módulo de Redondeo/-Saturación.
- **RF7 – Almacenamiento en FPGA:** Las imágenes de entrada y salida deben almacenarse en BRAM/M10K interna.
- **RF8 – Performance Counters:** El sistema debe registrar métricas como FLOPs, accesos de lectura/escritura y utilización de recursos.
- **RF9 – Stepping y Depuración:** Debe permitir ejecución paso a paso para observar estados intermedios de registros y memoria.
- **RF10 – Validación bit a bit:** El resultado de la interpolación hardware debe ser idéntico al modelo software de referencia.

3.2. Requisitos No Funcionales (RNF)

- **RNF1 – Plataforma:** El diseño debe ser completamente sintetizable para FPGA Cyclone V DE1-SoC MTL2.
- **RNF2 – Lenguaje de implementación:** El hardware debe desarrollarse en System-Verilog y el modelo de referencia en C/C++ o Python para validación.
- **RNF3 – Eficiencia:** Debe optimizar el uso de BRAM y DSP, manteniendo alta frecuencia de operación y bajo consumo.
- **RNF4 – Documentación y entregables:** Es obligatorio entregar código fuente, README, plan de verificación, bitstream y artículo científico en formato IEEE.
- **RNF5 – Trabajo en equipo y evaluación:** El desarrollo se realizará en grupos de tres integrantes con entregas semanales y demostración funcional obligatoria.

3.3. Matriz de Requisitos

ID	Descripción	Fuente en especificación	Criterio de aceptación
RF1	Implementa interpolación bilineal para downscaling de imágenes (8 bits).	Características generales (p. 1-2)	Simulación produce imagen escalada correcta.
RF3	Soporta modo secuencial y SIMD ($N \geq 4$).	Requisitos de arquitectura (p. 3-4)	Resultados hardware coinciden con modelo referencia.
RF5	Aritmética punto fijo Q8.8 en todas las operaciones.	Requisitos de software (p. 5)	Validación bit a bit entre HW y SW.
RF8	Contadores de rendimiento (FLOPs, R/W, intensidad).	Requisitos de arquitectura (p. 4)	Registro muestra métricas consistentes por modo.
RNF1	Síntesis en FPGA Cyclone V DE1-SoC.	Requisitos de hardware (p. 5)	Compilación y programación exitosa en Quartus.
RNF4	Entregables: código, README, plan de verificación, paper.	Entregables (p. 7-8)	Archivos entregados y verificados en TEC Digital.

Cuadro 2: Matriz de trazabilidad de requisitos del proyecto 02 (DSA para interpolación bilineal).

4. Marco conceptual

4.1. Interpolación bilineal

Dada una posición fraccional (x, y) en la imagen fuente, con $x_0 = \lfloor x \rfloor$, $y_0 = \lfloor y \rfloor$, $\alpha = x - x_0 \in [0, 1)$ y $\beta = y - y_0 \in [0, 1)$, y los cuatro vecinos $\{I_{00}, I_{10}, I_{01}, I_{11}\}$, el valor interpolado es:

$$I(x, y) = (1 - \alpha)(1 - \beta)I_{00} + \alpha(1 - \beta)I_{10} + (1 - \alpha)\beta I_{01} + \alpha\beta I_{11}.$$

Para **downscaling** por factor $s \leq 1$: $x = \frac{x'}{s}$, $y = \frac{y'}{s}$, con x', y' en la rejilla de salida.

Bordes. Se recomienda *clamp* de coordenadas (saturar al borde) o duplicación del borde.

Punto fijo. Con Q8.8 representamos coeficientes α, β y productos $w \cdot I$ sin FP. Sugerencia: acumular en 18–20 bits y redondear/saturar a 8 bits al final.

4.2. Paralelismo SIMD (DLP)

SIMD procesa múltiples datos con una misma instrucción. Para bilineal, un vector width $= N$ permite computar N píxeles de salida en paralelo siempre que podamos:

1. Cargar *ventanas* 2×2 por cada salida (deslizamiento horizontal/vertical).
2. Reutilizar vecinos entre salidas adyacentes (minimizando lecturas redundantes).
3. Vectorizar los cuatro productos ponderados y las sumas en **DSPs** en paralelo.

El *register file SIMD* guarda $\{I_{00}, I_{10}, I_{01}, I_{11}\}$ de N posiciones y los pesos α, β vectoriales; un *loader* por filas/columnas garantiza alineación y coalescencia sobre BRAM.

5. Diagrama de Arquitectura

La Figura 1 muestra el diagrama de primer nivel.

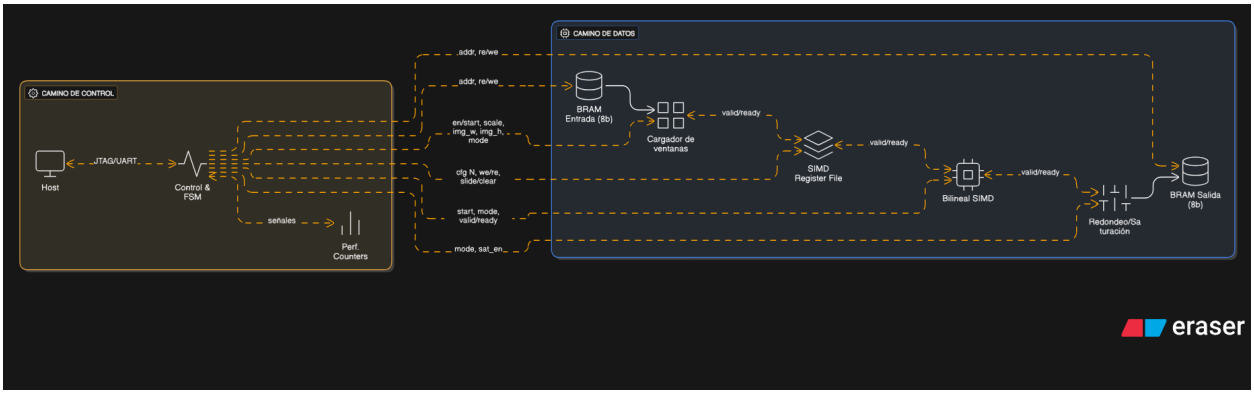

Figura 1: Arquitectura de primer nivel del DSA para downscaling bilineal.

6. Decisiones numéricas (Q8.8)

- Pesos α, β en Q0.8; píxeles se extienden a Q8.0 $\rightarrow$ productos en Q8.8.
- Acumulación de cuatro términos en Q10.8 (o mayor) para evitar *overflow*.
- Redondeo a entero (8 bits) y saturación $[0, 255]$ al escribir en BRAM de salida.

7. Plan de verificación (borrador)

1. **Unit tests** SV: loader (*stride*, bordes), registros SIMD, bilineal (vector escalar y N).
2. **Integración** secuencial: salida bit a bit igual al oráculo para escalas $s = \{0{,}95, 0{,}75, 0{,}5\}$.
3. **SIMD** $N \in \{4, 8\}$: equivalencia con secuencial, conteo de accesos y FLOPs.
4. **Stepping**: lectura de registros y estados en puntos de control predefinidos.

8. Algoritmo de alto nivel de downscaling

```
def downscale_bilinear_u8_to(img, out_w, out_h):

import math

def round_half_up(x):
    return int(math.floor(x + 0.5))

def clamp_u8(v):
    return 0 if v < 0 else (255 if v > 255 else v)

if not img:
    return []
H, W = len(img), len(img[0])
for row in img:
    if len(row) != W:
        raise ValueError("Non-rectangular image")

if out_w < 1 or out_h < 1:
    raise ValueError("out_w and out_h must be >= 1")
if out_w > W or out_h > H:
    raise ValueError("This function is for downscaling only (out_w<=W, out_h<=H)")

out = [[0] * out_w for _ in range(out_h)]

# Pixel-center mapping using per-axis scale
sx = W / float(out_w)   # > 1 for downscaling
sy = H / float(out_h)

for yd in range(out_h):
    y_src = (yd + 0.5) * sy - 0.5
    y0 = int(math.floor(y_src))
    fy = y_src - y0
    if y0 < 0:
        y0, fy = 0, 0.0
    y1 = y0 + 1
    if y1 >= H:
        y1, y0, fy = H - 1, H - 1, 0.0

    for xd in range(out_w):
        x_src = (xd + 0.5) * sx - 0.5
        x0 = int(math.floor(x_src))
        fx = x_src - x0
        if x0 < 0:
```

```
                x0, fx = 0, 0.0
            x1 = x0 + 1
            if x1 >= W:
                x1, x0, fx = W - 1, W - 1, 0.0

            I00 = img[y0][x0]; I10 = img[y0][x1]
            I01 = img[y1][x0]; I11 = img[y1][x1]

            # 3-multiply bilinear form
            a = I00 + fx * (I10 - I00)
            b = I01 + fx * (I11 - I01)
            v = a + fy * (b - a)

            out[yd][xd] = clamp_u8(round_half_up(v))

    return out
```

9. Conclusiones del avance

Se definió el marco teórico y la arquitectura base con los módulos críticos para vectorizar la bilineal en FPGA. El siguiente hito será cerrar el diagrama detallado (señales/anchos), fijar N y completar los testbenches del *loader* y la unidad bilineal.

Referencias

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Elsevier, 2017.

[2] Project F, "Fixed Point Numbers in Verilog," *Project F Blog*, 2025. [Online]. Available: https://projectf.io/posts/fixed-point-numbers-in-verilog/

[3] D. W. Gisselquist, "Strategies for Pipelining," *ZipCPU Blog*, 2017. [Online]. Available: https://zipcpu.com/blog/2017/08/14/strategies-for-pipelining.html

[4] Intel Corporation, "Intel 64 and IA-32 Architectures Optimization Reference Manual," 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/optimization-manual.html

[5] A. Benavides, "Guía JTAG para Comunicación FPGA-PC," *GitHub Repository*, 2025. [Online]. Available: https://github.com/Abner2111/GuiaJtag